



Command line and file editing hints and tips

Table of Contents

Introduction.....2
1. Always make a backup!2
2. Learn command line short cuts.....3
3. Use a text editor.....4
4) Learn to undo mistakes4
5) Editing/viewing large files.....5
6) Edit files using sed.....7
7) Be very careful when deleting files.....9
8) Always make backups!10
Conclusion.....10





Introduction

While Linux servers can run a desktop environment (e.g. KDE, Gnome), or use one of the many web-based server management systems (Webmin, cPanel, etc), it is still extremely useful to be able to edit files from the command line. You will often get a much clearer idea of how an application works by opening it's configuration file in a text editor, and viewing the various options listed within, than you will do by using a web-based, graphical interpretation of it's status and functions. However, when making those first steps onto the command line, it can seem like a scary and dangerous place. What follows in this paper are some tips to give you more confidence when editing files, and help you avoid making costly mistakes.

1. Always make a backup!

If you are going to edit a file, it's always a good idea to use the `cp` command to make a backup before making any changes. If the file is only a few lines long, this might seem excessive, but it can be a vital step in preventing downtime if your edit stops something from running - this can particularly be an issue when you're editing large configuration files - e.g. apache's `httpd.conf` - or files that depend on specific spacing/delineation for them to work (e.g. mailscanner's `filename.rules.conf`). Incorrectly editing either of these files can prevent the process from being restarted, and cause downtime while the mistake is located and corrected. If you have a backup of the original file, you can simply replace the edited file with the backup copy. For example, before editing `httpd.conf` run:

```
cp httpd.conf httpd.conf.BAK
```

Note: this assumes you're in the correct directory. If running the command from somewhere else within the directory structure, include the full path to the file, e.g.

```
cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.BAK
```

If editing the file causes problems, you can simply copy `httpd.conf.BAK` to `httpd.conf` and be back up and running without too much delay.





There are two switches that you might want to include when running this command:

-p = preserves the file attributes - ownership, permissions, etc.

-R = recursive copy everything within a directory.

So, if you were making a backup of a directory you might run something like this:

```
cp -pR /home/mysite /home/mysite.BAK
```

2. Learn command line short cuts

You can move around the shell environment if you learn a few simple short cuts. This can come in very handy if you're typing in long command strings directly onto the command line. Here's a few useful ones:

ctrl+a - move the cursor to the start of the current line.

ctrl+e - move the cursor to the end of the current line.

ctrl+w - delete the work to the left of the cursor.

ctrl+u - delete the current line.

NB: Press the ctrl (Control) key and the function key simultaneously; don't type out ctrl+a, etc, on the command line.

You can also use TAB to auto-complete commands and file names. Type the first letter(s) of the command/file/directory and press the tab key. If anything makes the letters, the shell will complete are much as it can. If the letters entered match several potential targets, pressing tab again will list the alternatives.

Finally, you might know that typing 'history' will display a list of commands you've previous run. But, if you want to re-run a command you've previously run, you can just type ctrl+r and you will be able to search back for a previous command. Just start typing the first bit of the command and the full command string will be displayed. For example, if you want to know what parameters you gave the 'cp' command, just type cp. You can use ctrl+p and ctrl+n to scroll up and down the full list of



previous commands in history.

3. Use a text editor

There are many different text editors available for Linux, but most sysadmins tend to use vi or vim - often abbreviated as vi(m) - although emacs is another favourite. However, while the basics are easy to learn, there are a huge selection of short cuts and functions available for each of these, and it can be quite easy to get confused and make mistakes as a result. Many people find nano is a good alternative. To open a file (let's call it test.txt) using nano, simply type:

```
nano test.txt
```

Unlike many of the other text editors, you don't have to enter an 'edit' or 'insert' mode to make changes to the file. Once opened, the text can be manipulated in the same manner as a notepad document, and you can freely scroll up and down the document. The bottom of the screen displays 12 control-keys, accessed by pressing ctrl and the letter assigned to the command, e.g. pressing ctrl+W opens a search box - type something in the box and press enter, and it will search for that text within the document. Pressing ctrl+O (letter O, not number 0) saves the current contents without exiting. Pressing ctrl+X will exit the current session, prompting you to save the file or exit without saving, if any changes have been made.

Nano is an enhanced, free clone of pico, the text editor used by the pine mail client. If you try to run pico on most systems, you will find it has been sym-linked to nano, and will actually open and run nano instead.

4) Learn to undo mistakes

If you do find yourself needing to use vi(m), get into the habit of running the process in the background while you test your changes. If you just save and exit using :qw, you won't be able to automatically undo any changes you've made using vi(m)'s undo command. To leave vi(m) open while checking the changes work, press :w to save the changes, but rather than exiting the program,

press ctrl+z to move the process into the background. You'll be dropped to a shell prompt, but if you run the command 'jobs', you will see the process is still running, e.g. If I ctrl+z while editing test.txt and then run jobs, I see this:

```
root@admin:~$ jobs
```

```
[1]+ Stopped          vim test.txt
```

When you've tested the changes, continue using vi(m) by bringing it into the foreground by typing 'fg'. This will reconnect you to your vi(m) session. If the changes need undoing, you can vi(m)'s udo command 'u' to remove the changes.

If, when you run jobs, there are several jobs running, e.g.

```
root@admin:~$ jobs
```

```
[1]- Stopped          vim test.txt
```

```
[2]+ Stopped          vim /etc/mysql/my.cnf
```

```
[3]+ Stopped          vim mybackup.csv
```

You can choose which job to reconnect to simply by typing the jobs number after typing fg. E.g. to continue editing you my.cnf file, you would type 'fg 2' then press enter.

5) Editing/viewing large files

Most text editors have to open the file being edited into memory, or into a temporary buffer, before you can read or edit the file. If the file is particularly large (some log files can easily take up several gigabytes) this can cause problems if there's not enough space in memory or the tmp partition.

For example, vi(m) will try and open the file into an editing buffer (/var/tmp, by default), and if there's not enough room on the partition it will return a "Not enough space in /var/tmp" error message. There are several options here, depending on how much overall space is available on the server. On average, vi(m) requires space equivalent to roughly twice the size of the file being edited to open it - the additional space is used for buffer manipulation. So, you can free up some space by



deleting files, or expand the size of the partition (if you're using LVM - Logical Volume Management). Or, you can change the directory vi(m) uses - to do this, open vi(m) by typing the command (vi or vim) without any options, e.g. In this example we will set vim to use /home and then open the file text.txt for editing by running the following commands:

```
vim                - opens vim
:set directory=/home - sets the directory to /home
:e test.txt        - opens test.txt for editing
```

Once opened, text.txt can be manipulated and saved by vim in the normal manner.

Another alternative is to use the split command to break the file up into smaller files, which can then be opened by vi(m). The standard syntax is: `split -l <lines> <filename>`

If our file text.txt is 100,000 lines long, we might want to split it into 20 files, each 5000 lines long. To do this we would run:

```
split -l 5000 test.txt
```

This will leave the original file intact but, if you run ls on the directory, you will now see 20 additional files - usually called something like xaa, xab, xac, etc - each of which will be 5000 lines long. These can then be opened using your preferred text editor.

If you just want to view a file, without editing it, use the 'less' command rather than a text editor, as this does not need to open the entire file before starting. This makes it faster to use, and avoids the memory/disk space issues associated with text editors. There are several switches that can be used with less, but to open a file the basic syntax is simply:

```
less test.txt
```

While you cannot edit the file using less, you can search the file by entering a / and then typing the search string and pressing enter.

Using less also prevents accidental changes being made to the file.

6) Edit files using sed

While sed (Stream EDitor) can be used to write fairly complex text manipulation scripts, even its most basic functions can be extremely useful. One of the most popular is the 'substitute' command, usually denoted by the letter 's' in at the start of a regular expression. This is used to find and replacing text strings within a file. Let's look at this example:

```
sed -i 's/old/new/' test.txt
```

Sed is invoked with the `-i` switch, which specifies the file should be edited 'in place'. If an extension is included (as it is in this case, we can see the file test has the extension txt) the extension is changed to create a temporary back copy of the file during editing. If no extension is included, the file is edited directly with no backup taken.

This is followed by a simple regular expression, which starts with the (s)ubstitute command and the strings (old and new) to be found and replaced are delimited by forward slashes. In this example, it simply means substitute the first occurrence of the word 'old' with the word 'new', followed by the file to be edited - test.txt.

Most Linux utilities will work on files one line at a time, and sed is no different, so this command will only substitute the first occurrence of the word 'old' on each line of the file. If you want to change every occurrence of a word, you must add the `g` (global) switch after the last delimiter (/), e.g.

```
sed -i 's/old/new/g' test.txt
```

This will now change every occurrence of the word old to new within the file test.txt.

If you have multiple files with the same extensions to edit, you can use wildcards for the file name. Let's assume you have several xml files, and you've noticed that all of them contain a spelling mistake - the word 'Eclipse' has been spelt as 'Esclipse'. You could open all of these files in a text editor and manually change this mistake, but it would be quicker and easier to run something like this:



```
sed -i 's/Esclipse/Eclipse/g' *.xml
```

This will look for every occurrence of the misspelt word within every xml document with the directory, and replace it with the correct spelling.

You can also use the -e (expression) switch to combine multiple expressions (you can also use ; as shown later), e.g.

```
sed -e 's/old/new/g' -e 's/day/night/g' test.txt
```

As well as replacing every occurrence of the word 'old' for 'new', this will simultaneously change every occurrence of the word 'day' for 'night'. You can also use a semi-colon to combine expressions together, e.g.

```
sed -e 's/old/new/g;s/day/night/g' test.txt
```

You can also use sed to delete words.

```
sed 's/hello//g' test.txt
```

This will replace every occurrence of the word 'hello' with nothing (as there's nothing between the second and third delimiters), effectively deleting it.

Sed is also useful for removing white space (spaces, tabs, etc). You can remove all leading and trailing white space by running:

```
sed 's/^[ \t]*//;s/[ \t]*$//' test.txt
```

Notice this is actually two statements connected by ; instead of the -e switch. The first half removes the leading white space (The ^ anchor matches the start of the string), with the second half removing trailing white space (the \$ anchor matching the end position).

You can also use sed to search a file using the -n switch, e.g. To search test.txt for occurrences of the word 'Key':

```
sed -n '/Key/p' test.txt
```

The -n switch filters out any lines that don't contain the keyword, and command p tells sed to print any matches to the screen.



There are many, many other uses for sed, but this should give you a taster for what can be done with even a few basic examples.

7) Be very careful when deleting files

The first point to make is, never delete anything unless you are sure you know what it is, and what it does. Deleting a configuration file, module or library file can potentially cripple your server if you delete something important. Always check and then double check before removing something and, if there's any doubt, don't remove it. It's sometime a better option to just rename and move a file somewhere else. Then, if problems arise, you can simply move the file back into place. The main tool for doing this is the mv command - which is used to move and rename files. The basic syntax is:

```
mv file1 file2
```

No additional paths have been specified, so all this example will do is rename file1 as file2 - it will stay within the same directory. To move it somewhere else you would sue:

```
mv file1 /home/sites/file2
```

The will move file1 from it's current directory to /home/sites and rename it file2.

If you need to remove a file, use the rm command. It's best to always use full path to the file (referred to as the 'absolute path'), rather than a relative path e.g.

Using an absolute path: `rm /home/sites/myfile`

Using a relative path: `rm sites/myfile`

The relative path example works fine if you're in the correct directory (home in this example), but can result in you deleting the wrong files/directories if you're in a different directory. It's always worth running the 'pwd' command (print working directory) to see which directory you are actually in, just in case you're not where you think you are. This especially true if you're using the -f switch, as this doesn't prompt you before removing the file(s)/directory, or the -r switch which recursively deletes everything within the targeted directory. For example, running something like `rm -rf /etc` as





root would delete the etc directory and all of its contents - as etc holds most of the system-wide configuration files, this would be very bad!

8) Always make backups!

Yes, we've already made this point, but it's worth making again. No other single piece of advice will save you so much time and effort as simply making backups of any file (especially configuration files) before you make changes to it.

Conclusion

This article has hopefully given you a taste of some of the things you can do on the command line, as well as providing you with some guidance towards avoiding some simple, but potentially disastrous, mistakes. There are entire books written about such things, but you can do a lot of further reading on the command line itself. Using the inbuilt man(ual) pages you can find out much more about the commands mentioned, along with further examples of their use and a complete list of switches that can be used with the command. Just type 'man' followed by the name of the command, e.g. to find out more about the cp (copy) command, type: man cp

